Algorytmy numeryczne

Zadanie 4 - Aproksymacja
Artur Pyśk & Aleksander Szewczak
Aplikacje internetowe i bazy danych

16 stycznia 2019

1 Aproksymacja

Celem naszego zadania było zastosowanie aproksymacji i weryfikacji jakości wyliczonej funkcji aproksymacyjnej dla następujących metod:

- Metoda Gaussa - wielomian 3-go stopnia,
- Metoda Gaussa z drobną optymalizacją dla macierzy rzadkich - wielomian 2-go stopnia,
- Metoda Gaussa-Seidla przy założonej dokładności 1e-10 - wielomian 2-go stopnia,
- Metoda zaimplementowana w oparciu o macierze rzadkie (SparseLU z biblioteki CSparse.NET) - wielomian 1-go stopnia,

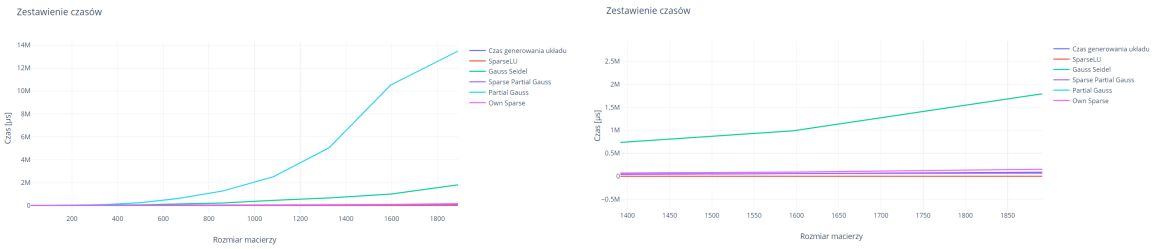
  Oraz:

  – Metoda z użyciem własnej struktury danych opisującej macierz rzadką - wielomian 2-go stopnia.

Program, który użyliśmy do analizy postawionego problemu został napisany w języku C#. Testy zostały wykonane na komputerze o następujących parametrach: Intel Core i5-6200U 2.4 Ghz, 8GB RAM, Windows 10.

2 Pomiary czasów

Wykonaliśmy pomiary czasu działania poszczególnych metod, stopniowo zwiększając rozmiar planszy. Aby uzyskać jak najdokładniejszy wynik, zmierzyliśmy czas działania każdej metody 100-krotnie, następnie odrzucaliśmy najbardziej skrajne czasy (min i max) i obliczaliśmy średnią pozostałych pomiarów.



(a) (b)

Wniosek 1 *Z załączonych powyżej wykresów wynika, że SparseLU z biblioteki CSparse.NET jest najszybszy. Zdecydowanie najwolniejszą metodą jest Partial Gauss. Zaimplementowana przez nas struktura jest znacznie szybsza od metody Gauss Seidel zgodnie z oczekiwaniami.*

3 Wielomiany aproksymacyjne

Dla każdej z metod wyznaczyliśmy funkcję aproksymacyjną w postaci wielomianu aproksymacyjnego o stopniu określonym w zadaniu.

PG: $12452.1589623475x^0 - 1917.22159190688x^1 + 3.70641805187391x^2 + 0.000654949588382414x^3$
Sparse PG: $-11.3635925692689x^0 - 2.45461511340257x^1 + 0.0210658033687397x^2$
Gauss Seidel: $2848.76130438716x^0 - 287.253779647719x^1 + 0.620711382874821x^2$
SparseLU: $-0.455552031292499x^0 + 0.0218823495216568x^1$
Own Sparse: $113.975736966916x^0 - 9.27471832487388x^1 + 0.0456211602424477x^2$

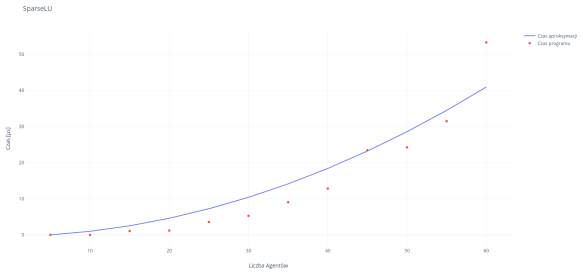
3.1 Poprawność implementacji

Poprawność implementacji potwierdza stworzony przez nas test jednostkowy w projekcie AlgNum-Projekt4Tests w klasie ApproximationTests o nazwie ApproximationResultTest. Został wykonany na podstawie przykładu udostępnionego z materiałów wykładowych (Prezentacja Aproksymacja, s. 34). Otrzymane przez nas wartości zgadzają się z przedstawionymi w materiałach.

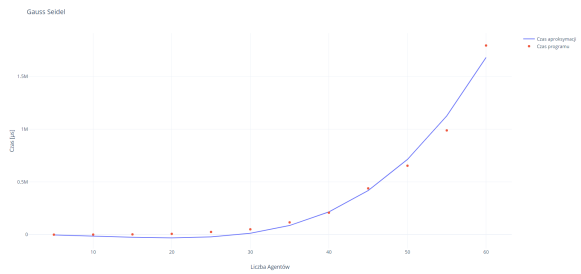
3.2 Błędy aproksymacyjne

Na poniższych wykresach przedstawiliśmy rzeczywisty czas działania funkcji z ich aproksymowanymi wartościami.
Szacowanie błędów aproksymacji przeprowadziliśmy na podstawie wzoru z wykładu:

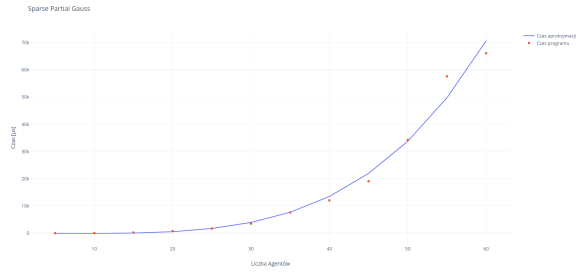
$$\|f(x) - F(x)\|_2 = \left(\sum_{i=0}^{n} w(x_i)(f(x_i) - F(x_i))^2 \right)^{1/2}$$



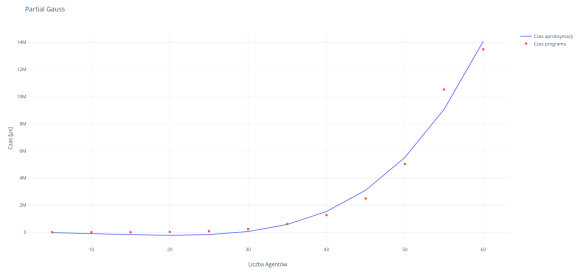
(a)



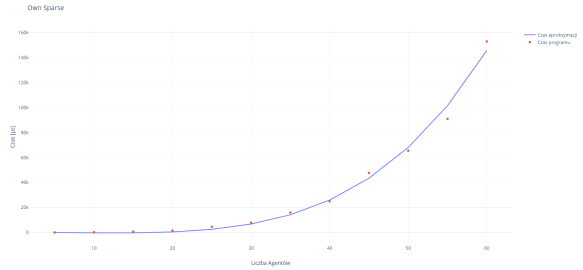
(b)



(c)



(d)



(e)

Wniosek 2 *Przedstawione na wykresach zestawienia potwierdzają poprawność uzyskanego przez nas rozwiązania.*

Wniosek 3 *Wszystkie metody z wyłączeniem SparseLU szacują czas w dosyć zadowalającym stopniu. Wynika to prawdopodobnie z zastosowania dodatkowych optymalizacji.*

3.3 Wartości błędów poszczególnych metod

Błąd	
Partial Gauss	4425372.53718232
Sparse Partial Gauss	18359.2949811507
Gauss Seidel	535645.725167602
SparseLU	45.2592450830897
Own Sparse	33380.2379920134

4 Ekstrapolacja

Poniższa tabela prezentuje czas, który wyliczono na podstawie funkcji aproksymującej dla liczby równań 100128.

Wyliczony czas aproksymacji	
Partial Gauss	691822.059194122 s
Sparse Partial Gauss	210.412560812464 s
Gauss Seidel	6178.39129954474 s
SparseLU	2187.77940013438 μs
Own Sparse	455.284244567726 s

Nie mogliśmy obliczyć problemu o wyznaczonym rozmiarze najszybszą metodą, którą była metoda SparseLU z powodu ograniczeń pamięciowych. Pomiar przeprowadziliśmy na największym możliwym rozmiarze - 54946.

Aproksymacja	1201 μs
Czas rzeczywisty	2856 μs

5 Własna struktura przechowywania macierzy rzadkich

Naszą własną strukturę opisującą macierz rzadką oparliśmy o system Compressed Row Storage, w implementacji składający się z następujących struktur:

- double[] Values - przechowujący wszystkie niezerowe elementy macierzy,
- int[] ColumnIndexes - przechowujący indeksy kolumn poszczególnych wartości, odpowiadającym indeksowo 1:1 ze struktury Values,
- int[] RowIndexes - przechowujący indeksy elementów ze struktury Values, w których rozpoczyna się kolejny rząd macierzy.

Zaimplementowaliśmy również zmodyfikowany proces generowania układu, który pozwolił nam na przeprowadzenie obliczeń dla 500 Agentów (rozmiar macierzy - 125751). Opiera się on na generowaniu wartości i automatycznym wychwytywaniu niezerowych elementów, które następnie są umieszczane w naszej strukturze, wraz z odpowiadającymi im indeksami kolumn oraz wyliczonym rzędem w strukturze Values. Dzięki temu obeszliśmy potrzebę budowania tablicy przekraczającą nasze zasoby, gdyż zużywamy pamięć tylko na niezerowe elementy i ich indeksy.

Poniżej przedstawiamy przykładową kompresję macierzy 6x6 przy $N = 2$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

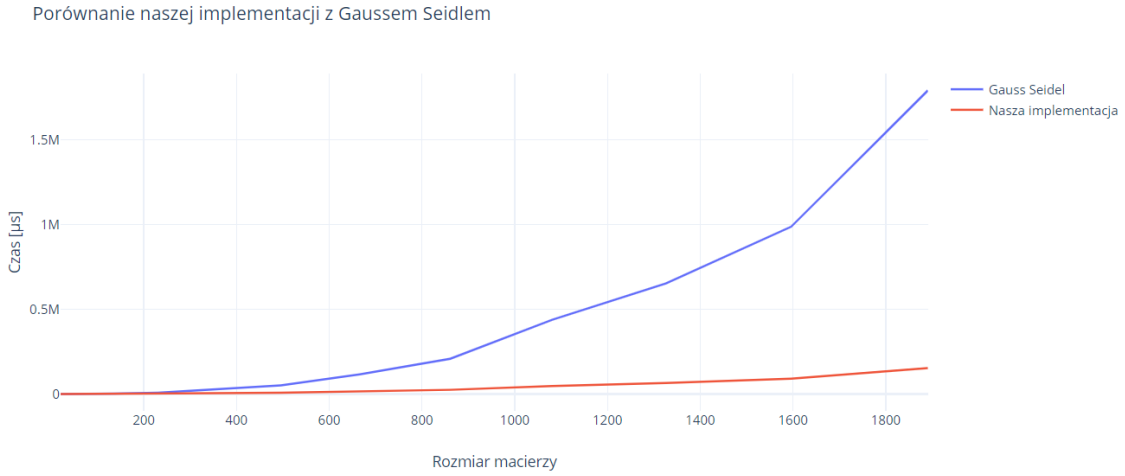
Values	1	-1	1	1	-1	1	1	-1	1
ColumnIndexes	0	1	2	2	3	5	0	4	5

RowIndexes	0	1	3	4	6	8

5.1 Poprawność implementacji

Naszą strukturę zaimplementowaliśmy w metodzie Gaussa Seidla, której czasy i otrzymane wartości porównywaliśmy z oryginalnie zaimlementowanym Gaussem Seidelem. Poprawność implementacji gwarantuje przygotowany test jednostkowy w projekcie AlgNum-Projekt4Tests, klasie EquationSolveTests i metodzie GaussSeidelSparseTest. Poprawność zweryfikowaliśmy poprzez sprawdzenie otrzymanego rozmiaru wektoru wynikowego, a także poprzez porównanie każdej wartości z wektora oryginalnego oraz otrzymanego przez naszą implementację.

5.2 Porównanie z Gaussem Seidlem



5.3 Wyniki dla 500 Agentów

Dzięki własnej implementacji udało nam się rozwiązać postawiony problem dla 500 Agentów. Program wykonywał się 66,38s.

6 Źródła

- Wykład Aproksymacja - dr inż. Łukasz Kuszner (dr inż. Piotr Borowiecki)
- `http://netlib.org/linalg/html_templates/node91.html`
- `https://github.com/wo80/CSparse.NET`
- `https://numerics.mathdotnet.com/api/MathNet.Numerics.LinearAlgebra.Double`

7 Podział pracy

Aleksander Szewczak	Artur Pyśk
Przeprowadzenie pomiarów czasów	Implementacja SparseLU
Aproksymacja	Sprawdzenie poprawności aproksymacji
Ekstrapolacja	Przeprowadzenie próby obliczenia problemu o wyznaczonym rozmiarze
Research własnej implementacji	Research własnej implementacji
Stworzenie własnej struktury	Implementacja własnej metody
Refactoring	Modyfikacja generowania dla własnej struktury
Testowanie	Pisanie testów jednostkowych
Pisanie sprawozdania	Pisanie sprawozdania